(19) BUNDESREPUBLIK DEUTSCHLAND

DEUTSCHES PATENT- UND MARKENAMT

(12) **Patentschrift**

(10) **DE 198 27 914 C 1**

(51) Int. Cl.[6]: **G 06 F 9/46**

(21) Aktenzeichen: 198 27 914.0-53
(22) Anmeldetag: 23. 6. 98
(43) Offenlegungstag: –
(45) Veröffentlichungstag der Patenterteilung: 28. 10. 99

Innerhalb von 3 Monaten nach Veröffentlichung der Erteilung kann Einspruch erhoben werden

(73) Patentinhaber:
Siemens AG, 80333 München, DE

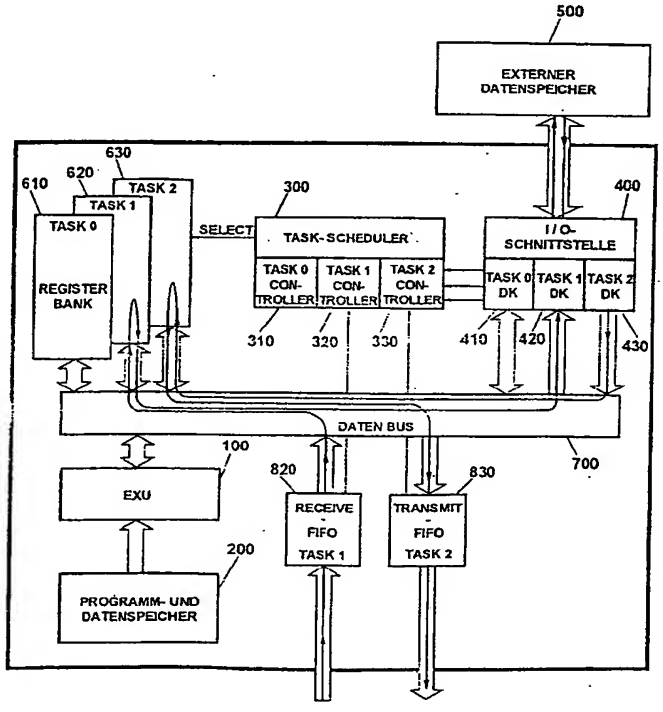
(72) Erfinder:
Osterholzer, Rudolf, Dipl.-Ing. (FH), 82008 Unterhaching, DE

(56) Für die Beurteilung der Patentfähigkeit in Betracht gezogene Druckschriften:

DE 195 30 483 A1
DE 43 01 117 A1
US 48 33 640

(54) Anwendungsspezifischer integrierter Schaltkreis mit einem RISC-Prozessor zur Bearbeitung definierter Sequenzen von Assembler Befehlen

(57) Die Erfindung bezieht sich auf einen ASIC zur Verarbeitung definierter Sequenzen von Assembler-Befehlen (TASKs). Zur Verbesserung des Datendurchsatzes bei Anwendungen mit hohen Speicherzugriffsraten, enthält der ASIC einen hardwareimplementierten TASK-Scheduler, der die Bearbeitung verschiedener TASKs auf einer ASIC internen Verarbeitungseinrichtung (EXU) in geeigneter Weise zeitlich koordiniert. Der hardwareimplementierte TASK-Scheduler bietet gegenüber herkömmlichen Softwaresteuerungen für Multitaskingsysteme unter anderem die Vorteile, daß das Betriebssystem entlastet und eine aufwendige Speicherarchitektur entbehrlich wird.



DE 198 27 914 C 1

Beschreibung

Die Erfindung bezieht sich auf einen anwendungsspezifischen integrierten Schaltkreis (ASIC) mit einem RISC-Prozessor zur Bearbeitung definierter Sequenzen von Assembler Befehlen (TASKs) gemäß dem Oberbegriff des Patentanspruchs 1.

Die Performance eines Prozessors ist stark von den Speicherzugriffszeiten abhängig, die erforderlich sind, um Programmcode oder Operanden in den Prozessor zu laden. Durch langsame Zugriffszeiten entstehen insbesondere in der EXU eines schnellgetakten RISC-Prozessors lange Totzeiten, die zwangsläufig zu einer Verschlechterung des Datendurchsatzes führen.

In der Vergangenheit wurden verschiedene Ansätze vorgeschlagen, um den Datendurchsatz eines Prozessors zu verbessern. Ein Ansatz besteht in der Implementierung aufwendiger Cache-Speichersysteme. Diese Cachesysteme sind allerdings für Anwendungen, bei denen die Daten nur ein einziges mal verwendet werden, sehr ineffektiv, weil dann eine hohe Cache-Miss-Rate entsteht.

Ein weiterer Ansatz zur Erhöhung des Datendurchsatzes bei I/O-Prozessoren besteht darin, bei längeren Wartezeiten während der Ausführung eines TASKs durch die EXU, einen software gesteuerten TASK-Wechsel durchzuführen. Dieser TASK-Wechsel wird folglich durch das Betriebssystem gesteuert. Er erfordert jedoch regelmäßig viele zusätzliche Speicheroperationen für das Sichern und Wiederherstellen des alten und neuen TASK-Zustandes, so daß sich ein derartiger Software-TASK-Wechsel nur bei sehr langen Wartezeiten, wie sie z. B. beim Zugriff auf externe Festplatten entstehen, lohnt.

Die US 4833640 offenbart ein Datenverarbeitungssystem, welches eine Vielzahl von Registerbänken in einem RAM aufweist. Der Zugriff auf die in diese Registerbänke und die in den Registerbänken enthaltenen Register erfolgt nach Maßgabe von entsprechenden Prozessorbefehlen. Der Befehl CBNR veranlaßt einen Wechsel des Zugriffs von einer Registerbank auf eine andere Registerbank bzw. einen Datentransfer von einem Register in der einen Registerbank in ein Register in der anderen Registerdatenbank. Die Registerzugriffe erfolgen unter der Steuerung eines Speicherzugriffssteuerschaltkreises und einer Vielzahl von Steuerregistern.

Die DE 43 01 117 A1 offenbart ein multitaskingfähiges Rechengerät und ein Verfahren zu seinem Betreiben. Das Rechengerät weist einen Prozessor mit mindestens einem Registersatz für jeden ausführenden Task auf. Das Rechengerät weist darüber hinaus einen Taskzuweiser auf, welcher dem Prozessor diejenige Task als nächste auszuführende Task zuteilt, welche die höchste Dringlichkeitsstufe aufweist.

Aus der DE 195 30 483 A1 ist eine Einrichtung und ein Verfahren zur Echtzeit-Verarbeitung einer Mehrzahl von Tasks bekannt. Die Abarbeitung der Tasks erfolgt durch einen Prozessor, wobei jeder Task nach einer dynamisch zugeordneten Priorität abgearbeitet wird.

Es ist die Aufgabe der Erfindung, einen anwendungsspezifischen integrierten Schaltkreis (ASIC) mit einem RISC-Prozessor mit einer verbesserten Auslastung seiner internen EXU bereitzustellen.

Diese Aufgabe wird durch den im Patentanspruch 1 beanspruchten Gegenstand gelöst.

Gemäß der Erfindung weist der RISC-Prozessor neben einem Programm- und Datenspeicher auch eine EXU zum Ausführen von TASKs auf, wobei während der Ausführung einer TASK auf eine andere TASK umgeschaltet werden kann, wenn zuvor die Ausführung des einen TASK unterbrochen wurde. Dabei wird der TASK-Umschaltvorgang durch einen hardwareimplementierten TASK-Scheduler gesteuert.

Die hardwaregesteuerte TASK-Umschaltung bietet den Vorteil, daß das Betriebssystem von der Aufgabe der TASK-Verwaltung entbunden ist, so daß es in verstärktem Maße andere Aufgaben erfüllen oder stark vereinfacht werden kann. Mit der Implementierung des TASK-Schedulers steht nun die gesamte Prozessorleistung der EXU ausschließlich zur Abwicklung von Anwenderfirmware zur Verfügung.

Der TASK-Scheduler ermöglicht zudem einen sehr schnellen TASK-Wechsel, d. h. innerhalb weniger Taktzyklen, wodurch die Totzeiten der EXU minimiert und ihre Auslastung bzw. ihr Datendurchsatz maximiert wird.

Gemäß der vorteilhaften Ausgestaltung des TASK-Schedulers weist dieser für jeden TASK einen eigenen TASK-Controller auf, der Veränderungen des aktuellen Zustandes (RUN, WAIT, READY oder HALT) des ihm zugeordneten TASK erkennt und steuert und eine entsprechende Veränderungsinformation erzeugt, die dem TASK-Scheduler zur Verfügung gestellt wird. Auf Basis diser Veränderungsinformation steuert der TASK-Scheduler den Umschaltvorgang.

Es ist weiterhin von Vorteil, in dem Schaltkreis eine I/O-Schnittstelle zwischen einem externen Datenspeicher und einem internen Datenbus vorzusehen, die für jeden TASK einen individuellen Datenkanal besitzt. Durch Überwachen des Datentransfers auf jedem einzelnen Datenkanal durch den individuell zugeordneten TASK-Controller kann eine Veränderung des Zustandes der jeweiligen TASK erkannt werden.

Als vorteilhaft sei außerdem erwähnt, daß die I/O-Schnittstelle (**400**) für den Datentransfer zwischen ihr und dem externen Datenspeicher (**500**) nur einen gemeinsamen Datenkanal für alle TASKs aufweist. Durch die Verwendung eines gemeinsamen Datenkanals für alle TASKs werden zwei Interfaces bzw. Datenkanäle eingespart.

Gemäß einer vorteilhaften Weiterbildung weist der Schaltkreis Registerbänke auf, die den TASKs individuell zugeordnet und jeweils an den internen Datenbus angeschlossen sind. Sie dienen zum Speichern von spezifischen Daten über den Zustand eines jeweiligen TASK zum Zeitpunkt seiner Unterbrechung oder von Daten, die von einem TASK vor seiner Unterbrechung bearbeitet wurden. Durch die Verwendung der Registerbänke entfällt die bei herkömmlichen Multitasking-Steuersystemen erforderliche komplexe Speicherarchitektur.

Gemäß einer besonders einfachen Ausgestaltung ist in dem Schaltkreis wenigstens ein FIFO-Speicher vorgesehen, der als unidirektionale Schnittstelle zwischen externen Komponenten und dem internen Datenbus angeordnet ist und durch einen ihm zugeordneten TASK-Controller ohne das Erfordernis einer aufwendigen Adressierung gesteuert wird.

Es folgt eine detaillierte Beschreibung eines bevorzugten Ausführungsbeispiels der Erfindung unter Bezugnahme auf die beigefügten Zeichnungen. Dabei zeigt:

Fig. 1 die Hardwarestruktur eines erfindungsgemäßen Schaltkreises; und

Fig. 2 ein Ablaufdiagramm für eine TASK-Umschaltung durch einen TASK-Scheduler.

Bevor nachfolgend der Schaltkreis gemäß **Fig.** 1 und das hardwaregesteuerte Umschalten einzelner TASKs gemäß **Fig.** 2 näher erläutert werden, erfolgen zunächst einige erläuternde Vorbemerkungen zum TASK-Begriff.

Wie bereits erwähnt, ist ein TASK ein Softwaremodul bestehend aus einer definierten Sequenz von Assembler Befehlen des verwendeten Prozessors. TASKs werden üblicherweise vom Programmentwickler als Module eines zu

erstellenden Programmcodes definiert. Bei vielen Anwendungsfällen ist eine Unterscheidung von drei TASKs ausreichend; genauer gesagt, werden ein Receive-TASK, ein Transmit-TASK und ein Supervisor-TASK unterschieden. Der Supervisor-TASK als Bestandteil des Programmcodes dient nur für die Initialisierung, zur Durchführung von Debug-Operationen sowie zur Fehlerbehandlung. Eine Beschreibung des Receive- und Transmit-TASK erfolgt weiter unten im Rahmen der Funktionsbeschreibung des Schaltkreises gemäß **Fig.** 1.

Die verschiedenen TASKs arbeiten i. d. R. unabhängig von einander, wenngleich gewisse Wechselwirkungen untereinander, wie gegenseitige Aufrufe oder eine Kommunikation über gemeinsame Speicherbereiche grundsätzlich möglich sind. Allerdings existiert eine TASK-Hierarchie, wobei der Supervisor-TASK die höchste Priorität besitzt. Aufgrund seiner höchsten Priorität kann der Supervisor-TASK die Receive- und Transmit-TASKs mit niederer Priorität jederzeit aufgrund eines Ereignisses unterbrechen.

Die Receive- und Transmit-TASKs können jeweils vier unterschiedliche Zustände einnehmen; diese sind:
der HALT-Zustand, bei dem der TASK inaktiv ist und von dem TASK-Scheduler **300** nicht beachtet wird,
der RUN-Zustand, bei dem der TASK aktiv ist und von der EXU **100** bearbeitet wird,
der WAIT-Zustand, bei dem der TASK zwar aktiv, aber blockiert ist, und
der READY-Zustand, bei dem der TASK ebenfalls aktiv ist, aber auf eine Zuteilung an die EXU wartet.

Der Supervisor-Mode kann ebenfalls die Zustände HALT, RUN und READY annehmen, nicht jedoch den WAIT-Zustand.

Der in **Fig.** 1 dargestellte Schaltkreis weist eine EXU **100** auf, die mit einem Programm- und Datenspeicher **200**, in dem definierte Sequenzen von Assembler Befehlen (TASKs) abgespeichert sind, verbunden ist. Über einen internen Datenbus **700** ist die EXU **100** mit einer I/O-Schnittstelle **400** verbunden, über die sie mit einem externen Datenspeicher **500** Daten austauschen kann. Die I/O-Schnittstelle **400** besitzt für den Datentransfer zwischen ihr und dem Datenbus **700** für jeden TASK, beispielsweise für einen TASK 0 (Supervisor-TASK), einen TASK **1** (Receive-TASK) und einen TASK **2** (Transmit-TASK) einen individuellen Datenkanal DK **410**, **420**, **430**, der jeweils mit dem Datenbus **700** verbunden ist. Die individuellen Datenkanäle **410** bis **430** sind jeweils über individuelle Steuerleitungen mit den ihnen jeweils zugeordneten TASK-Controllern **310** bis **330** verbunden. Für den Datentransfer zwischen ihr und dem externen Datenspeicher **500** weist die I/O-Schnittstelle lediglich einen gemeinsamen Datenkanal für alle TASKs auf. Dabei erfolgt der Übergang von drei auf einen Datenkanal innerhalb der I/O-Schnittstelle **400** durch Verwendung eines Multiplexers.

Die TASK-Controller sind Komponenten eines TASK-Schedulers **300**, welcher der EXU **100** jeweils einen TASK zur Bearbeitung zuweist.

Schließlich weist der Schaltkreis gemäß **Fig.** 1 für jeden TASK eine eigene Registerbank **610**, **620** bis **630** auf, die ebenso wie zwei FIFO-Speicher **820** (Receive-FIFO) und 830 (Transmit-FIFO) an den Datenbus **700** angeschlossen sind. Während die Registerbänke einen bidirektionalen Datenaustausch zulassen, gestattet jeder der FIFO-Speicher **820** und **830** lediglich einen unidirektionalen Datenaustausch mit dem internen Datenbus **700**. Der TASK-Scheduler **300** gibt über eine Select-Leitung jeweils diejenige Registerbank frei, die dem aktuell von der EXU **100** bearbeiteten TASK zugeordnet ist.

Es folgt nun eine Funktionsbeschreibung des Schaltkreises gemäß **Fig.** 1. Diese erfolgt am Beispiel der Abwicklung eines Receive- und eines Transmit-TASKs.

Der Receive-TASK wird hardwaregetriggert auf der EXU gestartet, wenn sein zugehöriger TASK-Controller **320** bei dem Receive-FIFO **820** den Eingang von Daten von externen Komponenten registriert. Die Ausführung des Receive-TASKs durch die EXU **100** bewirkt, daß die empfangenen Daten zunächst in die Receive-TASK-spezifische Registerbank **620** geladen werden, um dort modifiziert zu werden. Anschließend werden die modifizierten Daten über den Datenbus **700** und den Receive-TASK-spezifischen Datenkanal **420** der I/O-Schnittstelle **400** in den externen Datenspeicher **500** geschrieben.

Während der Receive-TASK auf der EXU **100** ausgeführt wird, befindet er sich im RUN-Zustand. Sollte jedoch im Rahmen der Ausführung des Receive-TASK festgestellt werden, daß keine Daten mehr im Receive-FIFO **820** vorhanden sind, so geht der Receive-TASK von selbst vom RUN-Zustand in den HALT-Zustand über.

Ganz ähnlich, allerdings mit entgegengesetzter Datentransferrichtung wie der Receive-TASK, arbeitet der Transmit-TASK. Er wird periodisch auf der EXU **100** gestartet, um Daten aus dem externen Datenspeicher **500** über seinen eigenen Transmit-Datenkanal **430** der I/O-Schnittstelle **400** und den Datenbus **700** in seine Registerbank **630** zu transferieren. Dort werden die Daten verarbeitet, um anschließend wiederum über den Datenbus in das Transmit-FIFO **830** geschrieben zu werden. Dort werden sie solange zwischengespeichert, bis sie an eine externe Komponente ausgegeben werden können. Während seiner Ausführung auf der EXU **100** befindet sich der Transmit-TASK im RUN-Zustand. Sollte während der Ausführung des Transmit-TASKs festgestellt werden, daß keine Daten mehr im externen Datenspeicher **500** zur Verarbeitung durch den Transmit-TASK anstehen, so geht auch der Transmit-TASK von selbst vom RUN-Zustand in den HALT-Zustand über.

Der Datentransfer auf den einzelnen Datenkanälen **410**, **420**, **430** wird permanent von den individuell zugeordneten TASK-Controllern **310**, **320**, **330** überwacht. Wenn ein Controller Veränderungen des Datentransfers auf 'seinem' Datenkanal feststellt, so lassen diese auf eine Veränderung des aktuellen Zustandes der ihm zugeordneten TASKs schließen, denn derartige Zustandswechsel erfolgen aufgrund von Hardwareereignissen, wie beispielsweise einem Zugriff auf den externen Datenspeicher **500**. Insbesondere kann ein einzelner TASK folgende Zustandswechsel durchlaufen:

Wechsel vom RUN- in den WAIT-Zustand (gilt nicht für Supervisor-TASK):

Solange der TASK von der EXU **100** bearbeitet wird, befindet sich der TASK im RUN-Zustand. Erfolgt jedoch im Verlauf der Bearbeitung über die I/O-Schnittstelle **400** ein Lesezugriff oder eine Schreiboperation auf den externen Datenspeicher **500**, so wickelt die I/O-Schnittstelle **400** mit ihren TASKspezifischen Datenkanälen **410**, **420**, **430** diese Operation selbsttätig, d. h. unabhängig von der EXU ab. Während dieser Zeit nimmt der TASK den WAIT-Zustand an. Bei einer Schreiboperation nimmt der TASK den WAIT-Zustand nur dann an, wenn unmittelbar davor ebenfalls eine Schreiboperation gestartet worden ist und diese noch den TASK-spezifischen Datenkanal belegt.

Wechsel vom WAIT-Zustand in den READY-Zustand (gilt nicht für Supervisor-TASK).:

Erfolgt während des WAIT-Zustandes eines TASK ein Lesezugriff auf den externen Datenspeicher **500**, und stehen

die angeforderten Lesedaten nun im TASK-spezifischen Datenkanal der I/O-Schnittstelle **400** zur Abholung durch den Prozessor bereit, so geht der TASK vom WAIT- in den READY-Zustand über.

Ein analoger Zustandswechsel erfolgt bei Schreiboperationen in den externen Datenspeicher **500**, wenn die einzuschreibenden Daten in dem TASK-spezifischen Datenkanal der frei gewordenen I/O-Schnittstelle **400** vor ihrem Transfer in den externen Datenspeicher **500** zwischengespeichert werden.

Wechsel vom READY-Zustand in den RUN-Zustand:

Ein TASK geht dann vom READY- in den RUN-Zustand über, wenn ihm Ausführungszeit der EXU zugeteilt wird, was nachfolgend ausführlicher anhand der **Fig.** 2 erläutert wird.

Wechsel vom RUN-Zustand in den HALT-Zustand:

Dieser Zustandswechsel erfolgt i. d. R. nur durch einen TASKeigenen Befehl, der bewirkt, daß der TASK inaktiv geschaltet wird. Er wird z. B. bei dem Receive- oder Transmit-TASK dadurch ausgelöst, daß keine weiteren Daten mehr zur Verarbeitung anstehen

Wechsel vom HALT- in den READY-Zustand:

Dieser Zustandswechsel geht einher mit einer Aktivschaltung eines zuvor inaktiven TASK. Eine Aktivschaltung kann z. B. durch ein Aktivierungssignal von einem Timer oder durch den Befehl eines anderen TASK erfolgen.

In der bisherigen Beschreibung erfolgte im wesentlichen eine Einzelbetrachtung der einzelnen TASKs. Insbesondere wurde ihre individuelle Funktionsweise beschrieben sowie ihre bei der Abwicklung durchlaufenen Zustandswechsel.

Aufbauend auf den von jeder TASK während ihrer Bearbeitung durchlaufenen Zustandswechsel, erfolgt nun anhand der **Fig.** 2 eine Beschreibung des durch den TASK-Scheduler **300** gesteuerten Multitaskings der verschiedenen TASKs.

In **Fig.** 2 sind für den TASK **1** und den TASK **2** beispielhaft Zustandswechsel dargestellt. So durchläuft der TASK **1** sukzessive die Zustände HALT READY RUN WAIT READY RUN, während der TASK **2** sukzessive die Zustände RUN WAIT READY RUN READY RUN WAIT READY durchläuft. Gleichzeitig durchläuft der TASK 0 die Zustände HALT READY RUN und HALT. Alle diese Zustandswechsel werden für jeden einzelnen TASK von dem ihm individuell zugeordneten TASK-Controller gesteuert.

Da nur eine EXU **100** vorhanden ist, ist auch nur die Abwicklung eines TASK zu gleicher Zeit möglich. Um die Auslastung der EXU und damit den Datendurchsatz durch den Schaltkreis zu optimieren, werden die einzelnen TASKs nicht etwa sequentiell, sondern nach dem Multitasking Prinzip durch die EXU abgearbeitet. Dabei erfolgt die für das Multitasking erforderliche Koordination der Zustandswechsel der einzelnen TASKs durch den hardwareimplementierten TASK-Scheduler **300**.

Wenn der TASK-Controller **330** einen Lesezugriff des TASK **2** auf den externen Datenspeicher **500** registriert, führt er den TASK **2** vom RUN- in den WAIT-Zustand über. Diese Veränderungsinformation teilt der TASK-Controller dem TASK-Scheduler **300** mit, welcher daraufhin den TASK-Controller **320** veranlaßt, den TASK **1** vom READY- in den RUN-Zustand zu überführen.

Damit wird die anfängliche Bearbeitung des TASK **2** auf der EXU unterbrochen und der TASK **1** der EXU zur Bearbeitung zugeteilt. Die Bearbeitung des TASK **1** auf der EXU erfolgt solange, bis der TASK-Controller **320** eine Schreiboperation des TASK **1** auf den externen Datenspeicher **500** registriert, woraufhin er den TASK **1** in den WAIT-Zustand überführt.

Solange nun, wie in **Fig.** 2 dargestellt, der WAIT-Zustand des TASK **1** noch nicht durch einen READY-Zustand ersetzt wurde, weil die Schreiboperation auf den externen Datenspeicher **500** noch nicht durch die I/O-Schnittstelle **400** beendet wurde, entsteht tatsächlich eine Totzeit für die EXU **100**, weil sich nun sowohl TASK **1** wie auch TASK **2** zeitgleich im WAIT-Zustand befinden.

Allerdings bleibt TASK **1** unabhängig von seinem Zustandswechsel vom RUN- in den WAIT-Zustand der EXU zugeordnet, wie dies aus der letzten Balkenzeile in **Fig.** 2 zu erkennen ist.

Ausgehend von dem gleichzeitigen WAIT-Zustand von TASK **1** und TASK **2** wird derjenige TASK als nächster von der EXU bearbeitet, dessen Operation mit dem externen Datenspeicher **500** als erste von der I/O-Schnittstelle erfolgreich abgewickelt und beendet wird. Sobald das Ende einer Operation als Veränderung des Datentransfers von einem TASK-Controller in einem Datenkanal DK der Schnittstelle erkannt wird, überführt er seinen TASK vom WAIT- in den READY-Zustand, was in der **Fig.** 2 für den TASK **2** dargestellt ist.

Der TASK-Controller **330** teilt daraufhin dem TASK-Scheduler **300** den READY-Zustand des TASK **2** mit, woraufhin dieser umgehend den TASK **2** der EXU zuteilt, weil TASK **1** nach wie vor im WAIT-Zustand ist. Mit der Zuteilung von TASK **2** zu der EXU **100**, überführt der TASK-Controller **330** seinen TASK **2** vom READY- in den RUN-Zustand.

Wie aus den Zustandswechseln in **Fig.** 2 für die TASKs **1** und **2** ersichtlich, wird durch das beschriebene Multitasking Prinzip die EXU **100** insgesamt besser ausgelastet, weil sie während des Wartezustandes eines zuvor im RUN-Zustand befindlichen TASK einen anderen TASK zur Bearbeitung zugeteilt bekommt.

Eine Unterbrechung des RUN-Zustandes eines TASK kann allerdings nicht nur durch Wartezeiten für eine externe Speicheroperation initiiert werden, sondern auch dann, wenn die EXU **100** von einem TASK höherer Priorität, z. B. dem Supervisor-TASK 0, beansprucht wird.

Dies geschieht üblicherweise dann, wenn dem TASK-Scheduler **300** vom TASK-Controller **310** der TASK 0 mitgeteilt wird, daß sich dieser im READY-Zustand befindet. Daraufhin unterbricht der TASK-Scheduler die Bearbeitung von TASKs mit niederer Priorität auf der EXU **100** und teilt dieser den TASK 0 zur Bearbeitung zu.

Daraufhin wird der TASK **2** von seinem Controller **330** aus dem RUN-Zustand in den READY-Zustand überführt, während gleichzeitig der TASK 0 von seinem Controller **310** vom READY- in den RUN-Zustand überführt wird.

Nach vollständiger Abwicklung des TASK 0, erteilt der TASK-Scheduler der EXU **100** wieder den zuletzt unterbrochenen TASK **2** zur Bearbeitung zu. Konsequenterweise wird dann der TASK 0 von seinem Controller **310** vom RUN- in den HALT-Zustand überführt, während gleichzeitig der zuvor unterbrochene TASK **2** von seinem Controller **330** wieder vom READY- in den RUN-Zustand überführt wird.

Solche Interrupts durch den Supervisor-TASK erfolgen allerdings üblicherweise nur bei besonderen Ereignissen, wie z. B. der Initialisierung, beim Auftreten von Fehlern, bei externer Alarmierung oder dann, wenn der Supervisor-TASK 0 von einem anderen TASK aufgerufen wird.

Jeder TASK für sich würde die EXU **100** nur etwa zur

Hälfte auslasten, weil die jeweilige Bearbeitung durch die EXU sehr häufig durch Zugriffe auf den externen Datenspeicher 500 unterbrochen werden müßte. Bei der Anwendung von TASKs, die jeweils etwa gleich viele Zugriffe auf den externen Datenspeicher **500** vorsehen, bietet der hardwareimplementierte TASK-Scheduler eine gute Möglichkeit, diese langsamen Zugriffe zeitlich zu verzahnen, um auf diese Weise den Datendurchsatz der Schaltung zu verbessern. Auch wenn der Prozessor durch die beiden TASKs sehr unsymmetrisch belastet wird, ergibt sich in der Gesamtbilanz für seinen Datendurchsatz immer noch ein Gewinn.

Patentansprüche

1. Anwendungsspezifischer integrierter Schaltkreis (ASIC) mit einem RISC-Prozessor zur Bearbeitung definierter Sequenzen von Assembler Befehlen (TASKs), mit einem Programm- und Datenspeicher (**200**) zum Speichern der TASKs, und einer Verarbeitungseinheit (EXU) (**100**) zum Ausführen der TASKs, wobei ein hardwareimplementierter Task-Scheduler (**300**) von einer TASK auf einen anderen TASK umschaltet, wenn zuvor die Ausführung des einen TASK unterbrochen wurde, **dadurch gekennzeichnet**, daß
jeder Task ein eigener Controller (**310, 320, 330**) zugeordnet ist, zum Erfassen des aktuellen Zustandes (RUN, WAIT, READY und HALT) der jeweils zugeordneten TASK und zum Ausgeben von entsprechender Zustandsinformation; und
daß der hardwareimplementierte TASK-Scheduler (**300**) so ausgestaltet ist, daß er die TASKs an die EXU (**100**) nach Maßgabe der erfaßten Zustandsinformation zuteilt.

2. Schaltkreis nach Anspruch 1, dadurch gekennzeichnet, daß eine I/O-Schnittstelle (**400**), zwischen einem externen Datenspeicher (**500**) und einem internen Datenbus (**700**) angeordnet ist.

3. Schaltkreis nach Anspruch 2, dadurch gekennzeichnet, daß die I/O-Schnittstelle (**400**) für den Datentransfer zwischen ihr und dem internen Datenbus (**700**) für jede TASK einen individuellen Datenkanal (**410, 420, 430**) besitzt, welcher den Datentransfer für jeden TASK durchführt, und daß Mittel vorgesehen sind, die den Datenkanal des zugeordneten TASK-Controllers (**310, 320, 330**) auf Veränderungen des Datentransfers hin überwacht, um aufgrund dieser Veränderung Veränderungen des Zustandes des jeweiligen TASK zu erkennen.

4. Schaltkreis nach Anspruch 2 oder 3, dadurch gekennzeichnet, daß die I/O-Schnittstelle (**400**) für den Datentransfer zwischen ihr und dem externen Datenspeicher (**500**) einen gemeinsamen Datenkanal für alle TASKs aufweist.

5. Schaltkreis nach einem der Ansprüche 2 bis 4, dadurch gekennzeichnet, daß Registerbänke (**610, 620, 630**), die den TASKs individuell zugeordnet und an den internen Datenbus (**700**) angeschlossen sind zum Speichern von spezifischen Daten über einen TASK nach einer Unterbrechung des jeweils ausgeführten TASKs, vorhanden sind.

6. Schaltkreis nach einem der Ansprüche 1 bis 5, dadurch gekennzeichnet, daß wenigstens ein FIFO-Speicher (**810, 820**) vorhanden ist, der als unidirektionale Schnittstelle zwischen externen Komponenten und dem internen Datenbus (**700**) angeordnet ist und durch einen ihm zugeordenten TASK-Controller (**320, 330**) gesteuert wird.

Hierzu 2 Seite(n) Zeichnungen

- Leerseite -

Nummer: DE 198 27 914 C1
Int. Cl.[6]: G 06 F 9/46
Veröffentlichungstag: 28. Oktober 1999

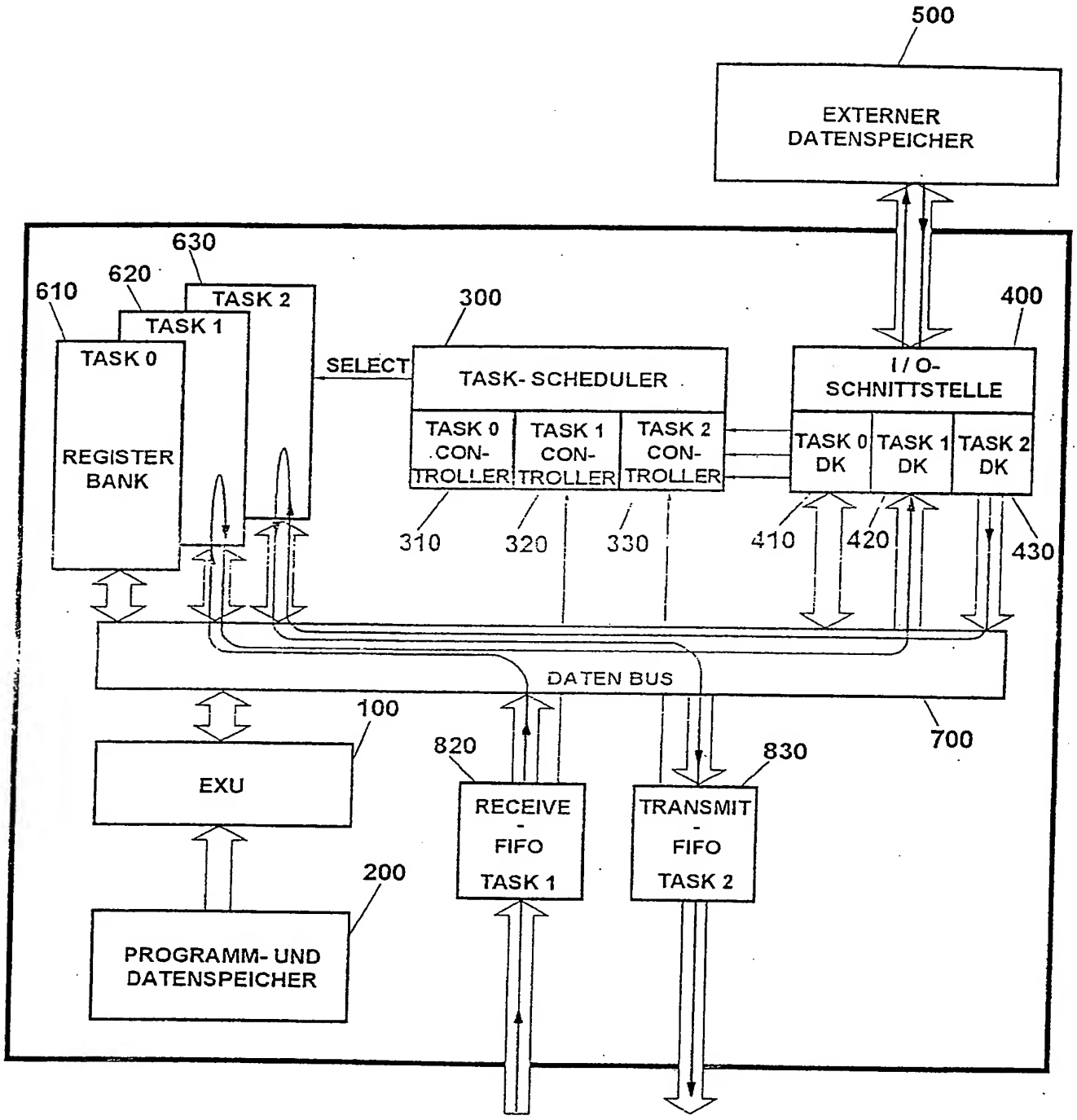


Fig. 1

Nummer: DE 198 27 914 C1
Int. Cl.[6]: G 06 F 9/46
Veröffentlichungstag: 28. Oktober 1999

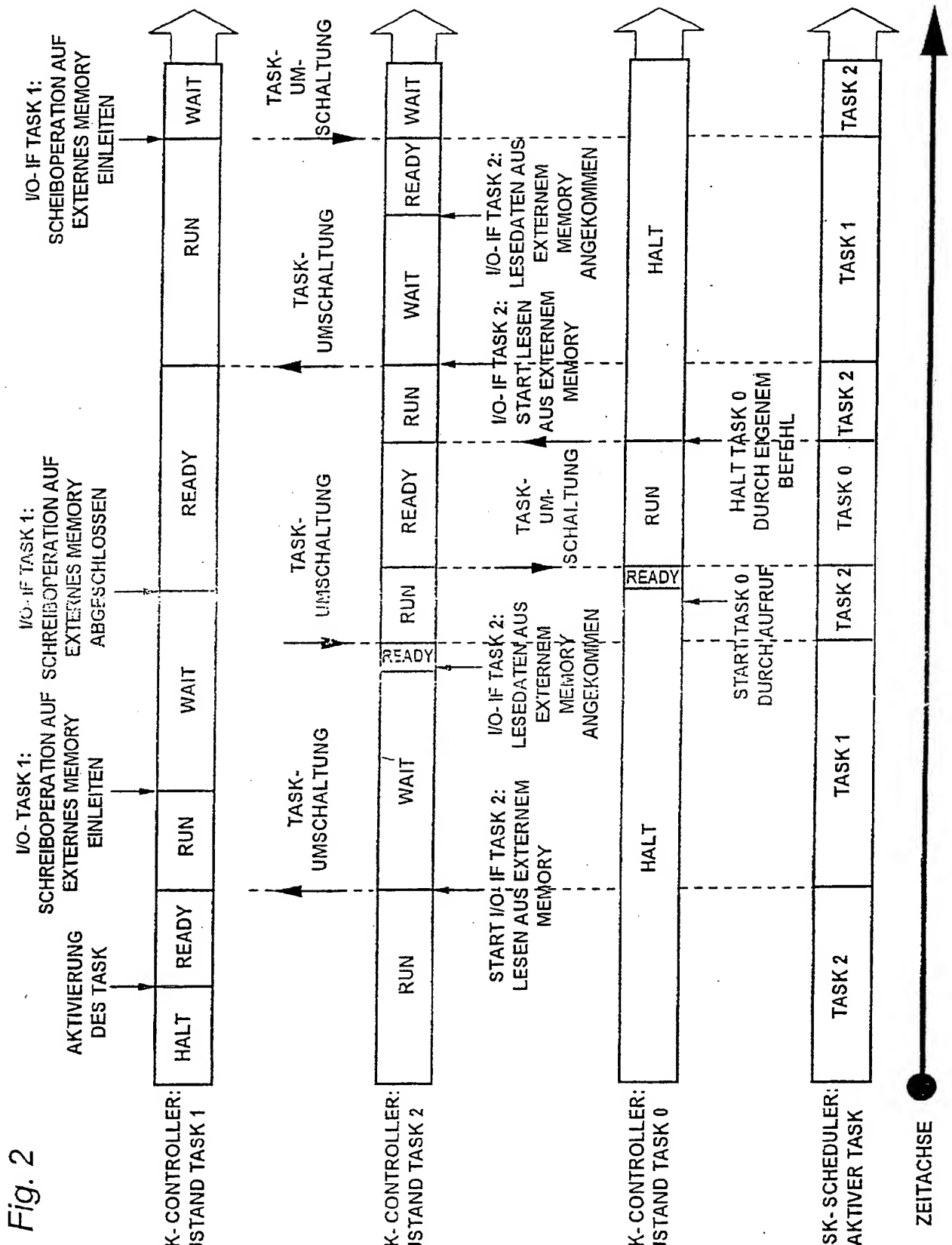


Fig. 2

902 143/286